Министерство образования и науки РФ

Белгородский государственный технологический университет им. В. Г. Шухова

Кафедра программного обеспечения вычислительной техники  
и автоматизированных систем

Курсовая работа

по дисциплине «Программирование на языке высокого уровня»  
тема «Создание программы для игры „Пентамино“»

Автор работы:  
студент группы ПВ-21  
Адаменко Игорь

Руководитель проекта:  
доцент  
Брусенцева В. С.

Белгород  
2013

# Содержание

[Введение 4](#_Toc374188554)

[Постановка задачи 5](#_Toc374188555)

[Выбор типа будущего приложения 6](#_Toc374188556)

[Консольное приложение 6](#_Toc374188557)

[Windows Forms 7](#_Toc374188558)

[Windows Store 7](#_Toc374188559)

[Графическая часть 9](#_Toc374188560)

[Выбор модели представления поля и фигур поля 9](#_Toc374188561)

[Изображения 9](#_Toc374188562)

[XAML 9](#_Toc374188563)

[Выбор способа взаимодействия с пользователем 9](#_Toc374188564)

[Клавиатура 10](#_Toc374188565)

[Джойстик 10](#_Toc374188566)

[Мышь, палец, стилус 10](#_Toc374188567)

[Связь графической части с программной 10](#_Toc374188568)

[Алгоритмическая часть 12](#_Toc374188569)

[Поведение фигуры на поле 12](#_Toc374188570)

[Перемещение 12](#_Toc374188571)

[Поворот 12](#_Toc374188572)

[«Прилипание» 12](#_Toc374188573)

[Выделение выбранной фигуры 12](#_Toc374188574)

[Выбор алгоритма нахождения решений 13](#_Toc374188575)

[Алгоритм полного перебора 13](#_Toc374188576)

[Алгоритм X Дональда Кнута 13](#_Toc374188577)

[Dancing links 17](#_Toc374188578)

[Реализация программы 19](#_Toc374188579)

[Графическая часть 19](#_Toc374188580)

[Поле 19](#_Toc374188581)

[Фигуры 19](#_Toc374188582)

[Алгоритмическая часть 19](#_Toc374188583)

[Хранение объектов в памяти 20](#_Toc374188584)

[Взаимодействия между объектами 20](#_Toc374188585)

[Нахождение решений 23](#_Toc374188586)

[Заключение 25](#_Toc374188587)

[Итоги работы 25](#_Toc374188588)

[Последующее применение программы 25](#_Toc374188589)

[Источники 27](#_Toc374188590)

[Литература 27](#_Toc374188591)

[Онлайн-ресурсы 27](#_Toc374188592)

[Приложения 30](#_Toc374188593)

[Скриншот меню приложения 30](#_Toc374188594)

[Скриншот игровой части 30](#_Toc374188595)

# Введение

Поскольку это курсовая работа по дисциплине «Программирование на языке высокого уровня», то для ее выполнения выбран один из них, а именно — C#.

C# — это прежде всего объектно-ориентированный язык, который разработан в 1998–2001 годах инженерами компании Microsoft, под руководством Андерса Хейлсберга, создателя таких языков программирования, как Turbo Pascal и Delphi. Он имеет C-подобный синтаксис, который наиболее близок к C++ и Java, а также имеет статическую типизацию, поддерживает полиморфизм, перегрузку операторов, делегаты, свойства, методы и все остальное, что присуще объектно-ориентированным языкам. Это делает его поистине языком высокого уровня.

Данный язык программирования многое перенял от своих предшественников — C++, Delphi, Java и других — и, опираясь на практику их использования, исключает при разработке некоторые проблематичные модели, как например множественное наследование классов.

Поскольку целью курсового работы является создание программы для игры «Пентамино», которая кроме запрограммированной логики должна включать в себя пользовательский интерфейс, C# является наилучшим инструментом для решения данной задачи, поскольку он разрабатывался специально для разработки приложений платформы Microsoft .NET Framework.

# Постановка задачи

Целью данной курсовой работы является написание программы для операционной системы Windows, которая позволит пользователю, запустившему ее сыграть в «Пентамино».

Пентамино — это игра-головоломка, целью которой является выложить из конечного числа фигур заданную форму. Пользователю дается некоторое количество пентамино (пятиклеточных плоских фигур), а также определенная форма (обычно прямоугольник), в котором ему необходимо уместить все заданные фигуры так, чтобы не осталось пустых мест и чтобы фигуры не пересекались между собой и не покрывали друг друга.

Также программа должна выдавать пользователю ответ по требованию. Причем, он должен находиться программой самостоятельно, анализируя начальный набор фигур и заданную форму.

Разработанная программа должна быть представлена в виде приложения для операционной системы Windows, имеющего пользовательский интерфейс.

Помимо реализации логики «Пентамино» приложение также должно запоминать, какие головоломки уже решены.

# Выбор типа будущего приложения

Перед тем, как разрабатывать приложение на C# под операционную систему Windows 8.1 стоит определиться с тем, как именно будет выглядеть будущее приложение. Из всех возможных типов можно выделить три основных:

1. Консольное приложение
2. Приложение с использованием Windows Forms
3. Приложение для Windows Store

Определим же основные плюсы и минусы использования каждого подхода, и какой из них больше подходит для реализации поставленной задачи.

## Консольное приложение

Это программа, которая для реализации своей визуальной части использует только консоль (командную строку, терминал), отсюда у нее есть следующие плюсы и минусы.

Плюсы:

* легко реализовывается, поскольку все управление в текстовом режиме и нет нужды в адаптации под определенные интерфейсы управления;
* запускается практически на всех современных операционных системах семейства Windows, а также на некоторых других (Linux, Mac OS X) при использовании специального программного обеспечения.

Минусы:

* пользователь, никогда ранее не работавший с командной строкой не разберется в управлении с первого раза;
* неудобное управление, поскольку в данной задаче нужно управлять положением фигуры на поле (т. е. пользователю придется вводить координаты фигуры вручную);
* данные могут вводиться только с клавиатуры или с ее эмуляторов (т. е. никакого взаимодействия с мышью);
* отсутствует возможность управления окном консоли и, как следствие, большие объемы данных могут теряться и быть неудобными для восприятия пользователем.

## Windows Forms

Windows Forms — это API[[1]](#footnote-1), отвечающий за GUI[[2]](#footnote-2) и являющийся частью платформы Microsoft .NET Framework. Его использование упрощает доступ к элементам интерфейса операционной системы Windows.

У использования Windows Forms так же есть свои плюсы и минусы.

Плюсы:

* более дружелюбный интерфейс, чем в консольном приложении;
* сравнительно простая разработка интерфейса приложения, за счет визуальных редакторов в IDE[[3]](#footnote-3);
* работает с мышью и прочими указывающими устройствами;
* запускается практически на всех современных операционных системах семейства Windows, а также на некоторых других (Linux, Mac OS X) при использовании специального программного обеспечения.

Минусы:

* несмотря на наличие графического интерфейса, полноценная реализация игрового приложения затруднительна.

## Windows Store

Вообще Windows Store — это название магазина Metro-приложений[[4]](#footnote-4) для Windows 8. Однако, поскольку единственный способ получить программы с Metro-интерфейсом — это скачать их из Windows Store, то такие приложения стали относить к одноименной категории.

Естественно, программа для Windows Store так же, как консольные приложения и Windows Forms имеет свои плюсы и минусы.

Плюсы:

* способность работать с любыми устройствами ввода;
* разворачивание в отдельное Metro-приложение, которое после малейших изменений также можно запускать на планшетах и современных телефонах на Windows Phone 8;
* гибкость работы с интерфейсом, реализовывающая весь возможный потенциал;
* использование для реализации интерфейса язык XAML[[5]](#footnote-5), что значительно упрощает работу с визуализацией.

Минусы:

* не работает на операционных системах не семейства Windows или версии ниже, чем 8.

Несмотря на существенный минус, разработка приложений для Windows Store — это то, что ждет разработчиков в ближайшем будущем, а потому, стоит выбрать именно этот тип для данной курсовой работы, чтобы в дальнейшем оказаться «на гребне волны», а также с легкостью охватить не только нишу настольных компьютеров и ноутбуков, но и смартфоны с ОС Windows Phone 8, а с ними и планшеты на Windows RT.

Именно из-за этого для решения поставленной задачи будет разрабатываться приложение для Windows Store.

# Графическая часть

Эта часть будущего приложения включает в себя все то, что видит пользователь. Учитывая тот факт, что приложения для Windows Store основаны на шаблоне программирования MVVM, в данном случае графическая часть — это View (Представление).

## Выбор модели представления поля и фигур поля

Исключая какие-то особо изощренные методы представления поля и фигур в приложении (использование сторонних графических библиотек и пр.), можно выделить два основных способа:

1. Представление в виде растровых изображений.
2. Представление с помощью встроенных средств XAML.

### Изображения

Этот способ подразумевает, что перед тем, как начать программировать алгоритм того или иного уровня игры, требуется воспользоваться графическими редакторами и создать растровое изображение (PNG, JPEG или др.) определенного фиксированного размера. Отдельно одно изображение для поля, и по изображению для каждой фигуры.

Однако у этого способа есть несколько существенных минусов:

* значительно возрастает размер скомпилированной программы, поскольку большую часть пакета составляют именно изображения;
* при редактировании уровня сперва потребуется изменить изображение, для чего необходимо снова воспользоваться графическим редактором.

### XAML

Использование разметки XAML и ее встроенных средств для «отрисовки» фигур более приоритетно в данном конкретном случае, поскольку это не только облегчит программу, но и поспособствует ее более быстрой работе, а также легкому редактированию фигур и полей.

## Выбор способа взаимодействия с пользователем

Количество средств управления объектом на экране компьютера (настольного, портативного, планшетного) ограничено, а потому можно выделить из них три основных:

1. Клавиатура.
2. Джойстик.
3. Указывающие устройства (мышь, палец, стилус).

### Клавиатура

Управление при помощи клавиатуры можно реализовать, выбирая фигуры нажатием соответствующих клавиш (F — фигура «F» и т. д.), их же перемещение возможно нажатием на клавиши-стрелочки. Однако этот способ не является удобным для пользователя.

### Джойстик

Управление с помощью джойстика (геймпада) тоже возможно, однако, оно ограничивает пользователей, поскольку у большинства из них нет подобных управляющих устройств.

### Мышь, палец, стилус

Это самый оптимальный вариант, поскольку если приложение запускается на планшетном или портативном компьютере, то управление возможно в любом случае, поскольку все они используют для этого или стилус, или палец, также большинство пользователей настольных компьютеров пользуются мышью, что является гарантом того, что они смогут воспользоваться данным приложением.

## Связь графической части с программной

Приложения для Windows Store реализованы при помощи шаблона MVVM, который реализует следующую архитектуру: Модель — Представление — Модель представления (Model — View — View-Model). Каждая из этих частей описывается следующим образом:

* Модель — фундаментальные данные, необходимые для работы приложения. В данном случае, это программная (алгоритмическая) часть данного приложения.
* Представление — это графический интерфейс, который «подписан» на изменения значений свойств или команд Модели представления. То есть, если в Модели представления изменилось какое-либо свойство, то она оповещает об этом всех подписчиков, а представление запрашивает у нее обновления. То есть, в случае взаимодействия пользователя с Представлением, последнее вызывает команду предоставленную Моделью представления.
* Модель представления — это, с одной стороны, абстракция Представления, а с другой — обертка данных для Модели, которые необходимо связать с Представлением.

Таким образом, графическая часть взаимодействует с программной посредством так называемого «связывания данных».

# Алгоритмическая часть

Несмотря на то, что основной задачей данной курсовой работы была реализация алгоритма поиска решения игры «Пентамино», для полноценной работы данного приложения потребовалось также реализовать алгоритмы управления фигурами на рабочем поле.

## Поведение фигуры на поле

Фигуры на рабочем поле можно перемещать и поворачивать. Также при перемещении в свободные ячейки поля они прикрепляются к нему, чтобы пользователю не приходилось самому их выравнивать.

### Перемещение

Реализовывается при помощи перехвата события *ManipulationDelta* над каждой из фигур. То есть, если пользователь задержал указатель на фигуре, а затем начал уводить его в определенном направлении («перетаскивать»), то следом за указателем меняется положение фигуры. После окончания перемещения, вызывается функция проверки ее текущего местоположения, и принимаются соответствующие манипуляции над данными (о них подробнее в разделе «Реализация программы»).

### Поворот

Реализовывается при помощи перехватов событий *RightTap* и *DoubleTap* над каждой из фигур. То есть, если пользователь дважды нажал пальцем (мышью) или кликнул правой клавишей мыши на фигуре, то она поворачивается на 90° вправо. Для фигур, у которых отсутствует потребность в таком повороте (о них подробнее в разделе «Реализация программы»), реализация поворота или отсутствует, или ограничена лишь несколькими положениями.

### «Прилипание»

Если после перемещения фигуры или ее поворота, она оказывается в таком месте игрового поля, на котором при небольшом ее [фигуры] отклонении от текущего положения она может попасть в незанятые ячейки — она перемещается в них.

### Выделение выбранной фигуры

При перемещении фигуры она становится прозрачной на 30%, чтобы пользователь явно видел, что он сейчас манипулирует ей. И если вдруг он оставит ее сверху какой-то другой фигуры на поле, то она останется прозрачной. Прозрачность фигуры устанавливается при возникновении события *ManipulationStarted* и возвращается (или не возвращается) к первоначальному при возникновении события *ManipulationCompleted*.

## Выбор алгоритма нахождения решений

Существует много различных подходов и алгоритмов для решения поставленной задачи. Отсекая все улучшения, можно выделить два основных:

1. Алгоритм полного перебора.
2. Алгоритм X Дональда Кнута.

Помимо этого также будет рассмотрено одно улучшение последнего алгоритма под названием «Dancing links».

### Алгоритм полного перебора

Заключается в применении поиска с возвращением к набору фигур и полю. Поле в данном случае представляется матрицей , где — количество ячеек поля по вертикали, а — по горизонтали. Фигуры также представляются матрицами, размеры которых не фиксированы, а зависят от размера конкретной фигуры. Матрица каждой из фигур заполнена уникальным символом, который позволяет идентифицировать фигуру.

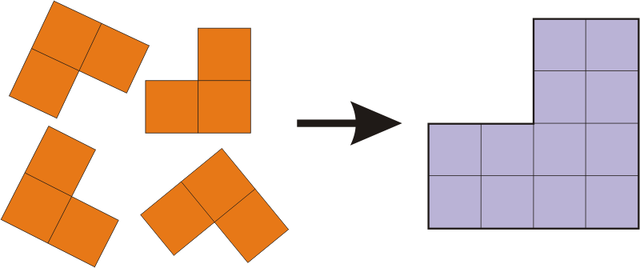
Суть алгоритма заключается в последовательном рекурсивном переборе фигур, которые поочередно вставляются в свободные места матрицы, а затем выполняется проверка. Работа алгоритма заканчивается, когда все возможные перестановки фигур (с учетом всех вариантов их положения на поле) были испробованы, или же при нахождении решения.

Однако, несмотря на тот факт, что алгоритм вполне рабочий, скорость его работы будет очень медленной, поскольку он не учитывает особенности фигур.

### Алгоритм X Дональда Кнута

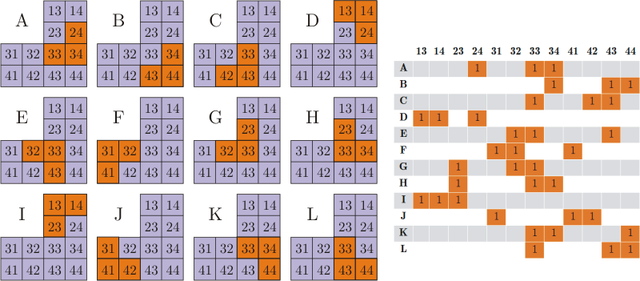
Фактически исходная задача является одним из типичных представителей задачи о точном покрытии, для решения которых существует «Алгоритм X», созданный Дональдом Кнутом. Чтобы продемонстрировать работу алгоритма, решим менее сложную задачу.

Представим, что у нас есть четыре одинаковые по размеру и форме фигуры из трех квадратиков в виде «уголка» (для простоты), которыми нам необходимо заполнить большую по размеру фигуру:

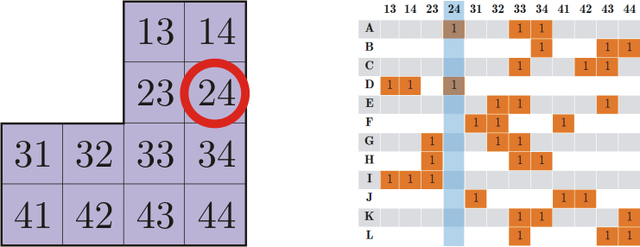


Пронумеруем все квадратики большой фигуры в виде , где — номер строки, а — номер столбца. Также пронумеруем все возможные положения маленькой фигуры внутри большой, используя для этого заглавные буквы латинского алфавита (ради простоты исключим изначально «плохие» положения, которые точно не приведут к решению). В большой фигуре мы имеем 12 квадратиков, которые нам и нужно покрыть положениями A–L.

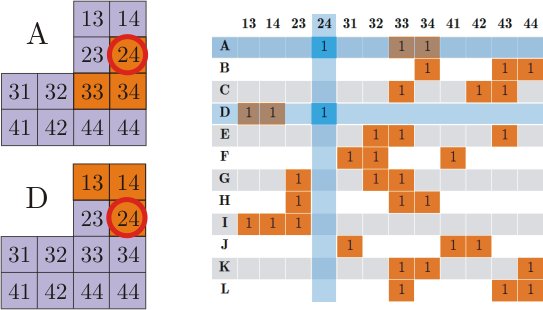
Эту задачу можно представить в виде матрицы инцидентности, в которой столбцы будут соответствовать квадратикам большой фигуры (т. н. ячейкам), а строки — положениям уголка в фигуре (т. е. буквам). Единица на пересечении строки и столбца ставится в том случае, если данное положение уголка покрывает данный квадратик.



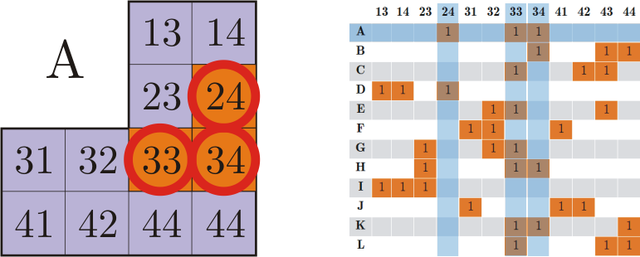
То есть в итоге решение задачи о покрытии уголками исходной фигуры сводится к тому, что нам требуется удалить из матрицы определенные строки, чтобы в итоге каждый столбец содержал ровно одну единицу. Распишем решение по шагам, обозначив матрицу за .



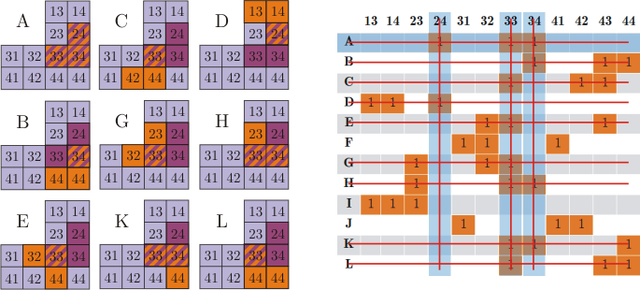
Шаг первый. Выбрать любой столбец в матрице . В данном случае, «выбор столбца» есть выбор какого-то непокрытого ранее квадратика. Допустим, что это квадратик с номером 24.



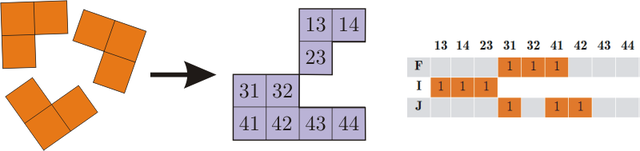
Шаг второй. Выбрать каждую строку , такую что . В данном случае квадратик 24 может закрываться как положением A, так и положением D. У нас нет каких-либо предпосылок к выбору того или иного варианта, поэтому мы рассматриваем каждый в отдельности. Для ускорения работы алгоритма, на первом шаге можно выбрать не произвольный столбец, а тот, в котором содержится наименьшее число единиц.



Шаг третий. Для выбранной строки рассматриваются все столбцы , такие что . В данном случае выберем строку A и внесем ее в стек решения. Теперь нам необходимо выделить все квадратики, которые покрывает положение A.



Шаг четвертый. Для всех ненулевых элементов столбцов удалим строки, в которых они находятся, а потом и сами столбцы . Все строки, которые имеют ненулевые элементы в выбранных столбцах, уже точно не могут участвовать в построении решения. Другими словами, мы избавляемся от всех положений уголка, которые пересекаются (т. е. имеют общие квадратики) с положением A. Более того, поскольку квадратики 24, 33, 34 мы уже покрыли (строка A уже находится в нашем стеке), то нам не нужно о них заботиться дальше, и соответствующие столбцы тоже удаляются.



Шаг пятый. Необходимо перейти к первому шагу, но уже с уменьшенной матрицей, которую мы получили после удаления строк и столбцов. Тем самым мы приходим к задаче о покрытии меньшей фигуры, причем, оставшиеся строчки автоматически соответствуют возможным положениям уголков в этой фигуре.

Окончание алгоритма. После перехода к первому шагу у нас могут возникнуть следующие особые ситуации:

1. Мы не можем выбрать столбец, потому что столбцов в матрице не осталось. Это означает, что мы покрыли все имевшиеся квадратики и, как следствие этого, нашли решение.
2. Мы выбрали столбец, но в нем нет ни одной единицы, как, например, у столбцов 43 и 44 на изображении выше. Это означает, что для данного квадратика нет фигур, которые могли бы его покрыть, а значит, эта ветвь не имеет решения. Частный случай такой ситуации — в матрице не осталось строк.

Поскольку за итерацию алгоритма мы заведомо избавляемся от одной строки и одного столбца, количество строк и столбцов постоянно уменьшается, и, рано или поздно мы придем или к первой, или ко второй особой ситуации.

### Dancing links

Для быстрой работы алгоритма необходимо как можно быстрее удалять строки и столбцы из матрицы. Если хранить матрицу в виде двумерного массива, то это достаточно проблематично (хотя в результате реализован именно такой подход; далее есть обоснование). С другой стороны, получаемая матрица практически всегда очень разряженная (в частности матрица этой задачи содержит всего примерно 8% единиц), и поэтому удобно представлять ее в виде двумерного двусвязного списка. Такое представление позволяет нам за минимальное время получать все ненулевые элементы строки или столбца. Наконец, имея в своем распоряжении такой список, очень удобно удалять и строки, и столбцы, а возвращаясь по ветке назад, восстанавливать их. Именно на этом основан «Метод танцующих ссылок» Дональда Кнута.

Однако эмпирическим путем было проверено, что, несмотря на то, что использование этого алгоритма дает увеличение скорости работы программы во время нахождения решения в 3 раза, это все равно недостаточно для того, чтобы пользователь мог «комфортно» работать с программой. Говоря ненаучным языком: программа все равно подвисает на 3-4 секунды.

Потому для наглядности решения выбрана именно стандартная реализация данного алгоритма.

# Реализация программы

Здесь разобраны и прокомментированы основные моменты программной части, как, например, причины выбора той или иной реализации алгоритмов.

## Графическая часть

Основными элементами графической части приложения являются Поле и Фигуры. Поле всего одно, фигур же двенадцать и все они разные. В Приложении можно посмотреть на скриншоты программы, на которых видно, как выглядят эти элементы.

### Поле

Реализовано целиком на XAML и представляет собой элемент *Grid,* то есть гибкую область сетки, состоящую из столбцов и строк. Однако само деление на клетки реализовано не встроенной сеткой, а множественным размещением элементов типа *Border,* поскольку так легче нарисовать именно «сетку поля», нежели при использовании встроенных разделителей. Обусловлено это тем, что для реализации границ между ячейками используется просто обрисовка самих ячеек, а не их разделение и фоновая подложка. Это позволяет более точно рассчитывать положение фигуры, так как каждая ячейка имеет фиксированный размер — 40×40 пикселей, что невозможно при использовании встроенных средств разметки элемента *Grid*.

Поле расположено по центру рабочей области (вертикально и горизонтально).

### Фигуры

Каждая фигура, как и поле, это *Grid,* в котором размещены элементы типа *Border*. Это тоже обусловлено тем, что так легче рассчитывать положение фигуры на рабочей области (и Поле в том числе).

Фигуры расположены на рабочей области слева и справа относительно Поля, что позволяет приложению стабильно отображаться как на маленьких, так и на больших разрешениях.

## Алгоритмическая часть

Разработка алгоритмов и структуры программы была направлена на последующее расширение функционала программы, что обусловливается алгоритмом, не привязанным к определенному размеру поля или набору фигур.

### Хранение объектов в памяти

Поле хранится в памяти как матрица , где — количество ячеек по вертикали, а — по горизонтали. Матрица поля передается в функцию-инициализатор во время запуска программы, тем самым определяя дальнейшее поведение фигур на поле.

Функция же, инициализирующая уровень и фигуры реализована в отдельном классе *(Data)*, что позволяет быстро добавлять в приложение новые задания, поскольку сам класс, его методы и свойства никак не привязаны к полю или набору фигур, используемых на текущем уровне.

Для хранения и связи объектов в классе *Data* реализованы следующие поля:

* gridToFigures — поле типа *Dictionary* (Словарь), являющееся таблицей, в которой хранятся записи вида *«Grid →* *Figures»*, где элемент типа *Grid* — XAML-фигура в рабочей области, а элемент структуры *Figures* — данные о фигуре (матрица, размеры, значение-заполнитель и пр.);
* currentLevel — матрица текущего уровня;
* solutionMatrix — матрица инцидентности, генерируемая при работе функции нахождения решений;
* foundedSolution — список-решение, содержащий матрицы с положением фигур;
* figure — поле типа *Grid,* содержащее информацию о текущей изменяемой XAML-фигуре на рабочей области;
* currentFigure — результат отображения *figure* на *gridToFigures;*
* field — поле типа *Grid,* содержащее информацию о XAML-объекте на рабочей области, символизирующем игровое поле.

Доступ к этим и другим, не описанным здесь полям в классе *Data* приватный, то есть невозможный извне. Манипуляции с этими полями производятся посредством вызова методов класса, которые, в свою очередь, вызываются событиями.

### Взаимодействия между объектами

*Важное примечание: Если в спецификации не указаны входные (выходные) параметры или возвращаемое значение, значит их нет.*

Спецификация статического метода InitializeFigures:

1. Заголовок: **public static void** InitializeFigures(Grid[] grids)
2. Назначение: Связывает программную реализацию фигур с графической, данные о которой содержатся в массиве *grids*
3. Входные параметры: grids

Спецификация статического метода InitializeLevel:

1. Заголовок: **public static void** InitializeLevel(Grid grid, **int**[,] level, TextBlock test)
2. Назначение: Инициализирует поля, хранящие данные о текущем уровне. *grid* — XAML-объект являющийся визуализацией игрового поля, *level* — матрица этого поля, *test* — XAML-объект (текстовое поле) для вывода информации при отладке (в Release-версии программы данный скрыт)
3. Входные параметры: grid, level, test

Спецификация статического метода InitializeFigure:

1. Заголовок: **private static void** InitializeFigure(Grid grid)
2. Назначение: Создает элемент в глобальном словаре *gridToFigures,* содержащий данные о фигуре, визуализацией которой является XAML-объект *grid*
3. Входные параметры: grid

Спецификация статического метода Motion:

1. Заголовок: **public static void** Motion(ManipulationDeltaRoutedEventArgs e)
2. Назначение: При возникновении события *ManipulationDelta* перемещает XAML-фигуру на рабочей области в соответствии с перемещением указателя (мыши, пальца, стилуса). *e* — данные события *ManipulationDelta*
3. Входные параметры: e

Спецификация статического метода Rotation:

1. Заголовок: **public static void** Rotation(**object** sender, RightTappedRoutedEventArgs e)
2. Назначение: При возникновении события *RightTapped* на XAML-фигуре (объект *sender*)*,* поворачивает ее на рабочем поле, одновременно с этим транспонируя ее матрицу, используя для этого данные о событии — *e*
3. Входные параметры: sender, e

Спецификация статического метода Swap:

1. Заголовок: **private static void** Swap(**ref int** first, **ref int** second)
2. Назначение: Обменивает значениями *first* и *second*
3. Входные параметры: first, second
4. Выходные параметры: first, second

Спецификация статического метода ToGrid:

1. Заголовок: **private static void** ToGrid()
2. Назначение: Проверяет текущее положение XAML-фигуры *figure* и, если она находится над игровым полем, то корректирует ее положение и «прикрепляет» ее к полю, если место для нее свободно.

Спецификация статического метода PasteOrCutFigure:

1. Заголовок: **private static void** PasteOrCutFigure(**bool** mode)
2. Назначение: Вставляет (или удаляет; в зависимости от *mode*) матрицу фигуры в матрицу поля.
3. Входные параметры: mode

Спецификация статического метода IsFreeZone:

1. Заголовок: **private static bool** IsFreeZone(**int** roundX, **int** roundY, **int**[,] mode)
2. Назначение: Проверяет, свободно ли место для фигуры, в матрице уровня *mode,* начиная с положения *[roundX, roundY]*
3. Входные параметры: roundX, roundY, mode
4. Возвращаемое значение: *true* — свободно, *false* — занято

Спецификация статического метода MotionOver:

1. Заголовок: **public static void** MotionOver()
2. Назначение: Вызывается при возникновении события *ManipulationCompleted,* для проверки текущего положения и вида (прозрачности) фигуры.

Спецификация статического метода RaiseZIndex:

1. Заголовок: **private static void** RaiseZIndex()
2. Назначение: Инкрементирует *ZIndex[[6]](#footnote-6)* текущей фигуры.

Спецификация статического метода MotionStarted:

1. Заголовок: **public static void** MotionStarted(**object** sender, ManipulationStartedRoutedEventArgs e)
2. Назначение: Проверяет положение XAML-фигуры (объект *sender*) и «вырезает» ее матрицу из матрицы игрового поля при начале ее перемещения, используя для этого информацию о событии *ManipulationStarted* — *e*
3. Входные параметры: sender, e

Спецификация статического метода Checker:

1. Заголовок: **private static bool** Checker()
2. Назначение: Проверяет, заполнена ли матрица текущего уровня
3. Входные параметры: —
4. Возвращаемое значение: *true,* если заполнена, иначе — *false*

Спецификация статического асинхронного метода MessageShow:

1. Заголовок: **private static async void** MessageShow(**string** header, **string** message)
2. Назначение: Отображает на экране всплывающее сообщение с заголовком *header* и телом *message*
3. Входные параметры: header, message

Спецификация метода ResetPage:

1. Заголовок: **private void** ResetPage(**object** sender, RoutedEventArgs e)
2. Назначение: При нажатии на кнопку сброса (XAML-объект *sender*) перезагружает текущую страницу (*e* — информация о событии *Click*)

### Нахождение решений

Спецификация статического метода RecursiveFinder:

1. Заголовок: **private static void** RecursiveFinder(**int** number, HashSet<**int**> notNeedRows, HashSet<**int**> notNeedCols, List<**int**[]> solution)
2. Назначение: Используя глобальный массив *solutionMatrix,* ищет при помощи Алгоритма X Дональда Кнута первое возможное решение задачи о покрытии игрового поля фигурами. *number* — номер текущей обрабатываемой строки, *notNeedRows* — множество индексов исключенных строк, *notNeedCols* — множество индексов исключенных столбцов, *solution* — список-решение с матрицами положений фигур
3. Входные параметры: number, notNeedRows, notNeedCols, solution

Спецификация статического метода FindAllPositions:

1. Заголовок: **private static int**[,] FindAllPositions(Figures figure)
2. Назначение: Генерирует и возвращает массив со всеми возможными положениями фигуры *figure* на игровом поле
3. Входные параметры: figure
4. Возвращаемое значение: целочисленная матрица, каждая строка которой — новое положение *figure,* а каждый столбец символизирует ячейку игрового поля

Спецификация статического метода BuildMatrix:

1. Заголовок: **private static int**[,] BuildMatrix()
2. Назначение: Генерирует матрицу инцидентности со всеми возможными положениями XAML-фигур на игровом поле
3. Входные параметры: —
4. Возвращаемое значение: целочисленная матрица, каждая строка которой — новое положение некоторой фигуры на поле, а каждый столбец символизирует ячейку игрового поля

Спецификация статического метода FindSolution:

1. Заголовок: **public static void** FindSolution()
2. Назначение: Ищет решение текущей головоломки

# Заключение

Выполненная курсовая работа существенно повысила мой уровень программирования на языке высокого уровня (в частности, на C#), поскольку пришлось использовать различные ранее не знакомые структуры данных (список, словарь, хеш-множество и пр.) и алгоритмы (Алгоритм X Дональда Кнута, Dancing links). Разработка велась под новейшую операционную систему, что, безусловно, дало неоценимый опыт, который впоследствии можно будет применить при создании более обширного программного обеспечения.

## Итоги работы

В результате курсовой работы был сделан вывод, что реализованное приложение (даже при максимальных улучшениях) не подходит для использования его в «игровых целях», поскольку вместо того, чтобы мгновенно получить ответ при запросе, пользователю приходится ждать, пока алгоритм закончит свою работу. Потому было решено применить реализованное приложение при разработке игры для массового использования.

В ходе работы также был реализован графический интерфейс для Windows Store приложения в классическом стиле Metro-приложений Windows. Он поддерживает управление мышью, пальцем или стилусом, а также запускается на Windows 8.1 и Windows RT (планшетная версия ОС). Выбор типа приложения, произведенный в начале разработки ОС, оправдан, поскольку теперь его легко портировать для Windows Phone (версия ОС для смартфонов), а также потому, что доля Windows 8 и Windows 8.1 на рынке операционных систем монотонно возрастает.

## Последующее применение программы

Благодаря реализации Алгоритма X Дональда Кнута, стала возможной быстрая и легкая разработка игры «Пентамино» для Магазина приложений Windows. С его помощью создаются новые уровни, поскольку алгоритм не зависит от вида задания, то используется для поиска ответа, который затем сохраняется в игре, что позволяет пользователю сразу получать ответ, а не ждать, пока алгоритм закончит работу.

Разработанная для широкой аудитории игра доступна в магазине приложений Windows (бесплатно) по следующему адресу: <http://apps.microsoft.com/windows/app/pentominoes/9cf45ae4-630e-44ce-bb1d-1cb90e907816>. Оно локализовано на русском, украинском, испанском и английском языках, и на момент написания курсовой работы уже имеет более 100 скачиваний, из которых скачивания из России составляют лишь 60%. Скриншоты игры доступны в разделе «Приложения».

Разработка приложения не останавливается, ибо благодаря алгоритмам, реализованным в данной курсовой работе, его функционал можно и нужно расширять.

# Источники

В данной курсовой работе использовались материалы из перечисленных ниже источников.

## Литература

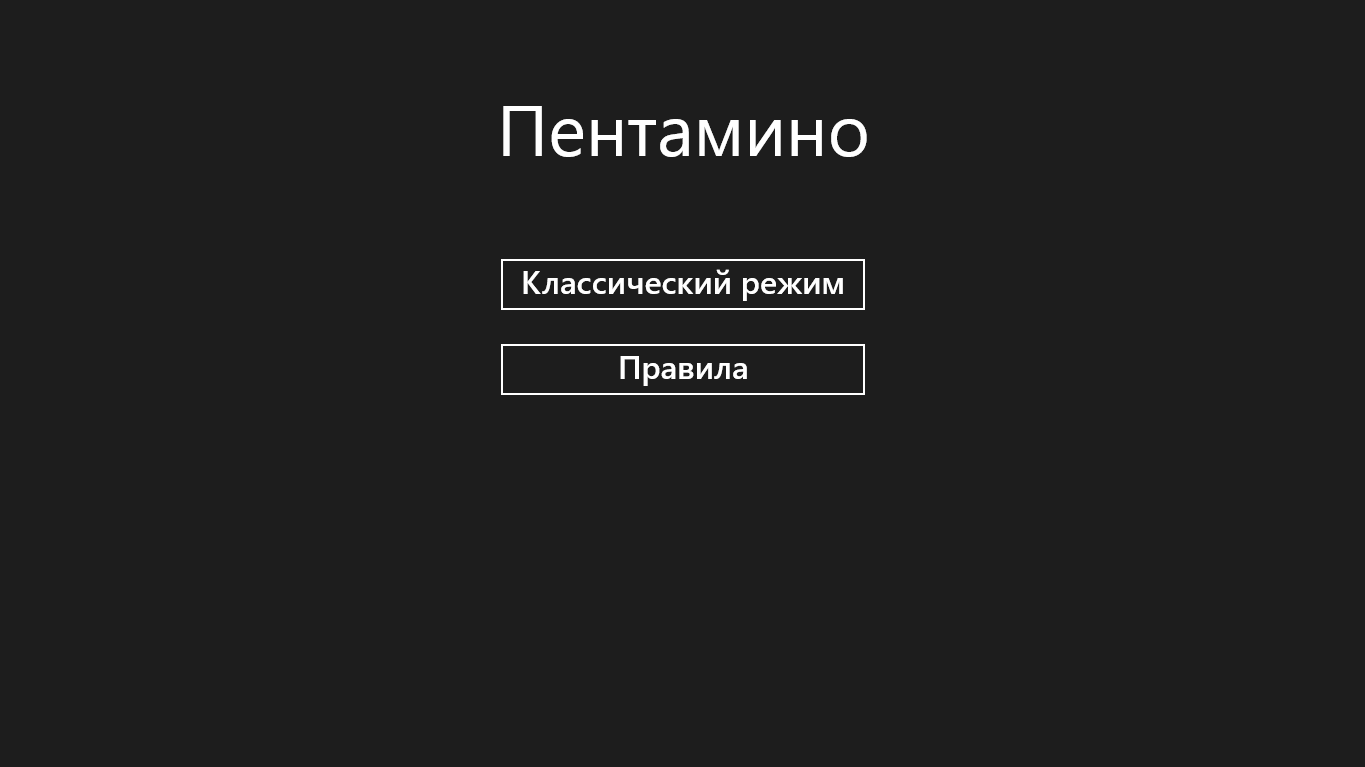
1. «Разработка приложений для Windows 8 на языке C#», С. В. Пугачёв, А. М. Шериев, К. А. Кичинский — СПб.: БХВ-Петербург, 2013. — 416 с.

## Онлайн-ресурсы

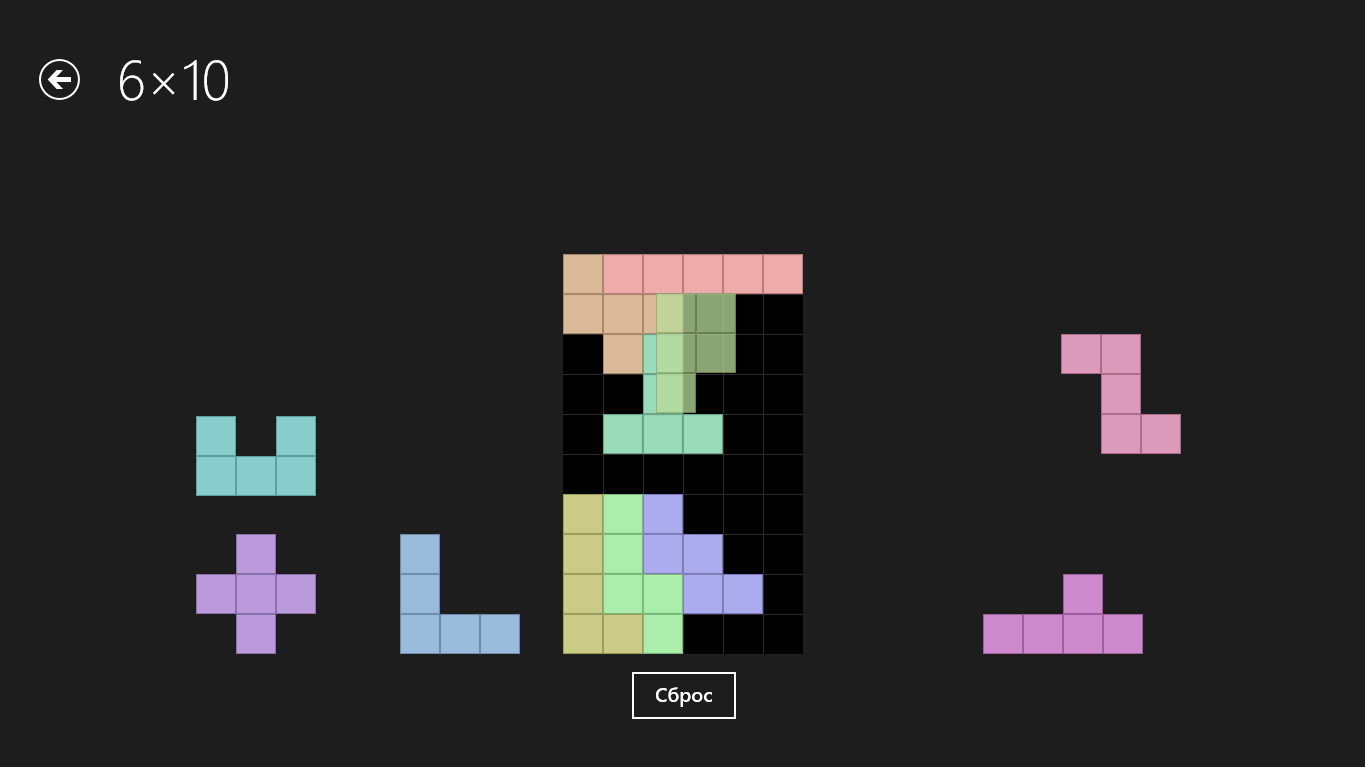
1. <http://msdn.microsoft.com/ru-ru/library/ms123401.aspx> — Библиотека MSDN
2. <http://stackoverflow.com/questions/12310731/adding-drag-and-drop-functionality-to-metro-style-app-buttons-with-c-sharp-and-x> — Adding drag and drop functionality to metro style app buttons with C# and XAML - Stack Overflow
3. <http://code.msdn.microsoft.com/Drag-and-Drop-a-picture-in-26580dc0> — Drag and Drop a picture in a Windows Store App in C#/XAML в C# для Visual Studio 2012
4. <http://stackoverflow.com/questions/12633512/how-do-i-drag-an-element-in-windows-8-xaml-app> — c# - How do I drag an element in Windows 8 XAML app? - Stack Overflow
5. <http://www.microsoftvirtualacademy.com/training-courses/app-dev-for-windows-8-xaml-c-sharp-rus> — Курс Разработка приложений для Windows 8 на XAML/C#
6. <http://stackoverflow.com/questions/19277239/how-to-build-pentomino-game> — c# - How to build Pentomino game - Stack Overflow
7. <http://stackoverflow.com/questions/12633512/how-do-i-drag-an-element-in-windows-8-xaml-app> — c# - How do I drag an element in Windows 8 XAML app? - Stack Overflow
8. <http://www.techdays.ru/videos/7175.html> — Новые возможности разработки на XAML/C# в Windows 8.1
9. <http://programming-pages.com/2012/01/15/wpfs-grid-layout-in-xaml-and-c/> — WPF’s Grid layout in XAML and C# | Programming tutorials
10. <http://stackoverflow.com/questions/19303656/how-to-group-shapes-in-xaml> — c# - How to group shapes in XAML? - Stack Overflow
11. <http://stackoverflow.com/questions/19306005/detect-grid-upon-grid-in-xaml> — windows 8 - Detect grid upon grid in XAML - Stack Overflow
12. <http://stackoverflow.com/questions/19305578/collapse-border-stroke-for-rectangles-in-xaml> — Collapse border (stroke) for rectangles in XAML - Stack Overflow
13. <http://stackoverflow.com/questions/19319371/find-out-that-user-has-finished-move-xaml-object-in-c-sharp> — wpf - Find out that user has finished move XAML-object in C# - Stack Overflow
14. <http://stackoverflow.com/questions/4132194/how-to-do-rotation-around-controls-center-in-xaml> — c# - How to do rotation around control&#39;s center in XAML - Stack Overflow
15. <http://stackoverflow.com/questions/19322390/coordinates-changed-after-rotation-xaml-shape> — c# - Coordinates changed after rotation XAML-shape - Stack Overflow
16. <http://stackoverflow.com/questions/19332131/change-canvas-zindex-for-grid-in-c-sharp> — xaml - Change Canvas.ZIndex for Grid in C# - Stack Overflow
17. <http://stackoverflow.com/questions/12349148/reload-page-in-metro-app-c-sharp> — xaml - Reload page in metro app C# - Stack Overflow
18. <http://habrahabr.ru/company/microsoft/blog/163711/> — Разбираемся с разработкой Windows 8 приложений на XAML/С#, реализуя простой RSS Reader. Ч.1 / Блог компании Microsoft / Хабрахабр
19. <http://diptimayapatra.wordpress.com/2012/09/05/message-dialog-in-windows-8-xaml-application-windows-store/> — Message Dialog in Windows 8 XAML Application (Windows Store) | Method ~ of ~ Tried
20. <http://stackoverflow.com/questions/9444071/how-to-rotate-a-image-in-xaml-c> — windows phone 7 - How to rotate a image in xaml/c# - Stack Overflow
21. <http://www.authorcode.com/how-to-rotate-shape-in-xaml/> — Transforming Shapes in XAML|how to transfrom shapes in wpf|How to rotate shape in wpf|How to rotate a triangle in wpf|transformation in Wpf | AuthorCode
22. <http://ru.wikipedia.org/wiki/XAML> — XAML — Википедия
23. <http://stackoverflow.com/questions/13000587/drag-and-drop-image-in-windows-8-metro-app> — c# - Drag and drop image in Windows 8 Metro App - Stack Overflow
24. <http://stackoverflow.com/questions/19315906/rotate-grid-around-cursor-position> — c# - Rotate Grid around cursor position? - Stack Overflow
25. <http://otvety.google.ru/otvety/thread?tid=01fc8ab593f1e07b> — как определить размер двумерного массива в C# - есть ответ - Вопросы и ответы
26. <http://csharp-algos.blogspot.ru/2010/10/merging-of-linked-lists.html> — Algorithms in C#: Merging of linked lists
27. <http://stackoverflow.com/questions/14983145/joining-two-2d-string-arrays> — c# - Joining Two 2D String Arrays - Stack Overflow
28. <http://stackoverflow.com/questions/7711130/concatenate-a-2d-array> — c# - Concatenate a 2D array - Stack Overflow
29. <http://stackoverflow.com/questions/59217/merging-two-arrays-in-net> — c# - Merging two arrays in .Net - Stack Overflow
30. <http://stackoverflow.com/questions/4622736/sortedsett-vs-hashsett> — c# - SortedSet<T> vs HashSet<T> - Stack Overflow
31. <http://answers.oreilly.com/topic/1949-how-to-use-the-hashset-and-sortedset-collection-classes-in-c-40/> — How to use the HashSet and SortedSet Collection Classes in C# 4.0 - O'Reilly Answers
32. <http://www.c-sharpcorner.com/UploadFile/0f68f2/comparative-analysis-of-list-hashset-and-sortedset/> — Comparative Analysis of List, HashSet and SortedSet
33. <http://stackoverflow.com/questions/1952185/how-do-i-copy-items-from-list-to-list-without-foreach> — c# - How do I copy items from list to list without foreach? - Stack Overflow
34. <http://stackoverflow.com/questions/4488054/merge-two-or-more-lists-into-one-in-c-sharp-net> — Merge two (or more) lists into one, in C# .NET - Stack Overflow
35. <http://stackoverflow.com/questions/4031262/how-to-merge-2-listt-with-removing-duplicate-values-in-c-sharp> — how to merge 2 List<T> with removing duplicate values in C# - Stack Overflow
36. <http://stackoverflow.com/questions/183685/c-sharp-set-collection> — .net - C# Set collection? - Stack Overflow
37. <http://habrahabr.ru/post/139734/> — C# 5 — об async/await с начала / Хабрахабр
38. <http://habrahabr.ru/post/162353/> — Использование async и await в C# — лучшие практики / Хабрахабр
39. <http://stackoverflow.com/questions/12435457/how-to-display-a-message-in-windows-store-apps> — c# - How to display a message in Windows Store Apps? - Stack Overflow
40. [http://ru.wikipedia.org/wiki/Пентамино](http://ru.wikipedia.org/wiki/%D0%9F%D0%B5%D0%BD%D1%82%D0%B0%D0%BC%D0%B8%D0%BD%D0%BE) — Пентамино — Википедия
41. <http://en.wikipedia.org/wiki/Pentomino> — Pentomino - Wikipedia, the free encyclopedia
42. <http://en.wikipedia.org/wiki/Exact_cover> — Exact cover - Wikipedia, the free encyclopedia
43. <http://en.wikipedia.org/wiki/Algorithm_X> — Knuth's Algorithm X - Wikipedia, the free encyclopedia
44. <http://habrahabr.ru/post/194410/> — Алгоритм Х или что общего между деревянной головоломкой и танцующим Линком? / Хабрахабр

# Приложения

## Скриншот меню приложения



## Скриншот игровой части



1. API (англ. *application programming interface*) — интерфейс программирования приложений; набор готовых классов, функций, структур и констант, предоставляемых библиотекой или сервисом (в данном случае ОС Windows) для использования их в других, внешних продуктах. [↑](#footnote-ref-1)
2. GUI (англ. *graphical user interface*) — графический интерфейс пользователя. [↑](#footnote-ref-2)
3. IDE (англ. *integrated development environment*) — интегрированная среда разработки. [↑](#footnote-ref-3)
4. Приставка «Metro» пошла от первоначального названия оболочки Windows 8. Также используется название «Modern UI». [↑](#footnote-ref-4)
5. XAML (англ. *extensible application markup language*) — язык разметки для декларативного программирования приложений, разработанный Microsoft. [↑](#footnote-ref-5)
6. ZIndex — свойство XAML-объекта, отвечающее за то, насколько близко он находится к пользователю по оси Z (чем больше значение, тем ближе); в данном случае используется для того, чтобы фигура, над которой производятся манипуляции, находилась выше всех остальных и не перекрывалась ими. [↑](#footnote-ref-6)